93 C# interview questions to hire top developers

Questions

1. What is C# and why do we use it?

2. Explain the difference between value types and reference types in C#.

3. What is the purpose of the 'using' statement?

4. Describe the concept of inheritance in C#.

5. What are interfaces in C# and how are they different from abstract classes?

6. What is polymorphism and how is it achieved in C#?

7. Explain the difference between '==' and '.Equals()' in C#.

8. What is a delegate in C#?

9. What are events in C# and how are they used?

10. Describe the purpose of exception handling in C#.

11. What is LINQ and what are its benefits?

12. What is an extension method and how do you create one?

13. Explain the difference between 'const' and 'readonly' keywords.

14. What are generics in C# and why are they useful?

15. Describe the purpose of attributes in C#.

16. What is boxing and unboxing in C#?

17. Explain the difference between 'as' and 'is' operators.

18. What is the purpose of the 'sealed' keyword?

19. Describe the difference between stack and heap memory.

20. What are nullable types in C# and why are they useful?

21. Explain how garbage collection works in C#.

22. What is asynchronous programming in C# and when should you use it?

23. Describe the purpose of the 'virtual' and 'override' keywords.

24. What is the difference between a struct and a class?

25. Explain the concept of lambda expressions in C#.

26. What are collection initializers in C#?

27. Explain the difference between `Func<T, TResult>` and `Action<T>` in C#.

28. What are extension methods, and how can they be used? Provide an example.

29. How does the `yield` keyword work in C#, and what problem does it solve?

30. What is the purpose of the `async` and `await` keywords in C#, and how do they work together?

31. Describe the difference between `Task.Run()` and `Task.Factory.StartNew()`.

32. What is LINQ, and how does it improve code readability and maintainability? Give a basic example.

33. Explain the concept of deferred execution in LINQ.

34. Describe the differences between `IEnumerable` and `IQueryable`.

35. What are the benefits of using interfaces in C#? Explain with an example.

36. Explain the difference between `struct` and `class` in C#.

37. When would you choose to use a `struct` over a `class` in C#?

38. What is boxing and unboxing in C#, and what are the performance implications?

39. How does the garbage collector work in C#? What is the difference between generations?

40. Explain the purpose of the `using` statement in C# and how it relates to the `IDisposable` interface.

41. What are delegates in C#, and how are they used to implement event handling?

42. Explain the difference between delegates and events in C#.

43. Describe how you would implement a custom exception in C#.

44. What are attributes in C#, and how can you create and use them?

45. Explain what reflection is in C#, and provide a scenario where it might be useful.

46. What are generics in C#, and how do they improve type safety and performance?

47. Explain the differences between `async` and `await` and how they contribute to building responsive applications.

48. How does the .NET garbage collector work, and what are the different generations of garbage collection?

49. Describe the purpose of the `IDisposable` interface and the `using` statement in C#, and explain how they help manage resources.

50. What are delegates and events in C#, and how do they facilitate communication between objects?

51. Explain the concept of LINQ (Language Integrated Query) and how it simplifies data querying in C#.

52. What are extension methods, and how can they be used to add functionality to existing classes without modifying their source code?

53. Describe the differences between value types and reference types in C#, and how they affect memory management.

54. What is reflection in C#, and how can it be used to inspect and manipulate types at runtime?

55. Explain the purpose of attributes in C# and how they can be used to add metadata to code elements.

56. What are generics in C#, and how do they enable type-safe programming with reusable code?

57. How does the `yield` keyword work in C#, and how can it be used to create iterators?

58. Describe the concept of covariance and contravariance in C# generics, and provide examples of their usage.

59. What are tuples in C#, and how do they provide a way to group multiple values into a single object?

60. Explain the purpose of the `dynamic` keyword in C#, and how it enables late-bound programming.

61. What are lambda expressions in C#, and how can they be used to create anonymous functions?

62. Describe the different types of collections available in C#, such as lists, dictionaries, and sets, and explain their use cases.

63. What is dependency injection (DI) in C#, and how can it be used to improve the testability and maintainability of code?

64. Explain the concept of the Task Parallel Library (TPL) in C#, and how it simplifies parallel programming.

65. What are asynchronous streams in C#, and how do they enable the processing of data streams asynchronously?

66. Describe the purpose of the `HttpClient` class in C#, and how it can be used to make HTTP requests.

67. What are custom attributes, and how would you implement and use them for custom metadata handling in your applications?

68. Explain different ways to handle errors and exceptions in C#, including try-catch blocks, custom exceptions, and global exception handling.

69. Can you discuss the purpose and benefits of using immutable data structures in C# for concurrent programming?

70. How does C# support interoperability with unmanaged code, and what are the challenges associated with it?

71. Explain the concepts of multi-threading and parallelism in C#, and how would you implement them to improve performance?

72. Explain the nuances of using `async` and `await` in a complex C# application. How do you ensure proper error handling and prevent deadlocks?

73. Describe scenarios where you would use a custom `TaskScheduler` in C#. Explain how it differs from the default scheduler and the benefits it provides.

74. How does the C# compiler handle closures, and what are the potential pitfalls you should be aware of when using them extensively?

75. Discuss the trade-offs between using structs and classes in C#, focusing on memory allocation, performance, and potential boxing/unboxing issues.

76. Explain the internals of the C# garbage collector. How can you profile and optimize your code to reduce garbage collection pressure?

77. Describe the implementation details of LINQ deferred execution. How does it affect performance and debugging, and how can you optimize LINQ queries?

78. What are the advantages and disadvantages of using immutable data structures in C#? How do you effectively implement them and handle updates?

79. Explain the use cases for different types of collections in C# (e.g., `ConcurrentDictionary`, `ImmutableList`). When should you prefer one over another?

80. How does the C# runtime handle dynamic method invocation? What are the performance implications and use cases for dynamic programming in C#?

81. Discuss the design patterns applicable for building a scalable and maintainable microservices architecture using C# and .NET.

82. Explain how you would implement a custom attribute in C# to enforce specific coding standards or perform compile-time validation.

83. Describe the different ways to implement inter-process communication (IPC) in C#. What are the trade-offs of each approach?

84. How would you optimize a C# application for high-throughput and low-latency scenarios? Consider threading, memory management, and network communication.

85. Explain the advanced features of C# delegates, such as multicast delegates and covariance/contravariance. Provide use case examples.

86. Describe the implementation and usage of custom iterators in C#. How do they differ from standard `IEnumerable` implementations?

87. Explain the concept of `Span<T>` and `Memory<T>` in C#. How do they improve performance when working with memory buffers?

88. How would you design a C# application to be resilient to transient faults in a distributed environment? Consider using Polly or similar libraries.

89. Discuss the various ways to serialize and deserialize objects in C#. What are the performance and security considerations for each method?

90. Explain how you would implement a custom diagnostic source and listener in C# to monitor and diagnose performance issues in a production environment.

91. Describe the role of Roslyn analyzers and code fixes in improving code quality. How can you create custom analyzers for your team's coding standards?

92. How would you implement a generic retry mechanism in C# that handles different types of exceptions and implements exponential backoff?

93. Explain how you can leverage C# features like source generators to automate repetitive tasks and reduce boilerplate code in your projects.